Computers Math. Applic. Vol. 22, No. 6, pp. 1–11, 1991
Printed in Great Britain.

0097-4943/91 $3.00 + 0.00

THE CLIQUE-PARTITIONING PROBLEM*

J. BHASKER†
AT&T Bell Laboratories
1247 S. Cedar Crest Blvd., Allentown, PA 18103
TARIQ SAMAD
Honeywell, Corporate Systems Development Division
1000 Boone Avenue North, Golden Valley, MN 55427

(Received May 1990)

Abstract—We study the problem of clique-partitioning a graph. We prove a new general upper bound result on the number of clique-partitions. This upper bound is the best possible, given information of just the vertices and the number of edges. Next we show that there exists an optimal partition in which one of the cliques is a maximal clique. Finally we present two new efficient methods to clique-partition a graph. Since the clique-partitioning of a graph is equivalent to the coloring of the complement of the graph, any coloring algorithm can also be used to clique-partition a graph. We present detailed statistics comparing the performance of our algorithms against two of the best known coloring algorithms and against a recently published clique-partitioning algorithm. Both functional and timing comparisons are given and we show that our algorithms compare very favorably on both counts.

1. INTRODUCTION

A *clique* of a simple graph $G = (V, E)$, where V is the set of vertices in the graph and E the set of edges, is a subset W of V such that for every pair of vertices in W, there is an edge in E. The *clique-partitioning* of a graph refers to the problem of finding the smallest number of cliques in a graph such that every vertex in the graph is represented in exactly one clique. The *coloring* of a graph refers to the problem of finding the smallest set of "colors," such that each vertex of the graph is assigned exactly one color from this set, and that no two vertices that are connected have the same color. The clique-partitioning of a graph G is equivalent to the coloring of the complement graph G'.

Both clique-partitioning and coloring have applications in a variety of fields. Thus good, efficient algorithms for these problems as well as theoretical results on, for example, upper and lower bounds and complexity, are of significant importance.

In this paper, we study the problem of clique-partitioning a graph. We first prove a new general upper bound result on the number of clique-partitions. This upper bound is the best possible, given information of just the vertices and the number of edges. Next we show that there exists an optimal partition in which one of the cliques is a maximal clique. Finally, we present two new efficient methods to clique-partition a graph, one of which is an extension of the other. Since the clique-partition of a graph is equivalent to the coloring of the complement of the graph, any coloring algorithm can also be used to clique-partition a graph. We briefly review some well-known coloring algorithms and a recently published clique-partitioning algorithm and we present

*This paper results from work done on Contract No. F33615-85-C-1861 from Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Division (AFSC), United States Air Force, Wright-Patterson AFB, Ohio 45433–6543.
†This work was performed by the author while he was employed at Honeywell Corporate Systems Development Division, Golden Valley, MN 55427.
We thank Tammy Kappel for her help with the simulations.

the results of some detailed comparative experiments. Both functional and timing comparisons are given and we show that our algorithms compare very favorably on both counts.

We use θ to represent the minimum number of cliques into which a graph can be partitioned. The upper bound on θ is symbolized as θ_{upper}. n represents the number of vertices in the graph and e the number of edges. The minimum degree of any vertex in the graph is represented by δ, and the maximum degree by Δ. χ is the minimum number of colors needed to color a graph. Other quantities that we will use in the sequel is the size of the largest clique in the graph, referred to as γ, and the size of the maximum independent set of a graph, referred to as α.

2. A BETTER UPPER BOUND

We now present a new upper bound for clique-partitioning. This upper bound applies to all simple graphs, and is optimal in the sense that a better upper bound cannot be derived given just n and e.

The upper bound hinges on an intuition: In the clique-partitioning of a graph, at least one of the possible edges between each pair of cliques must not be present otherwise that pair of cliques can be consolidated into one clique. (By an edge between two cliques we mean an edge between any vertex in one clique and any vertex in the other.)

For a graph to have c cliques, then the following condition must hold:

$$e \leq \frac{n(n-1)}{2} - \frac{c(c-1)}{2}$$

or,

$$c^2 - c - n(n-1) + 2e \leq 0$$

Solving for c, we get

$$c \leq \frac{1 + \sqrt{4n^2 - 4n - 8e + 1}}{2}$$

Thus we get the following theorem.

THEOREM 1.

$$\theta_{\text{upper}} = \left\lfloor \frac{1 + \sqrt{4n^2 - 4n - 8e + 1}}{2} \right\rfloor \tag{1}$$

Equation (1) is in fact an *optimal* upper bound. That is, given just the information about the number of vertices and edges of a graph, we cannot derive a lower upper bound than expressed in Equation (1). To prove this, we show that for every n and e, we can construct a graph that has exactly θ_{upper} cliques, as given by Equation (1).

Let n be the number of vertices in the graph, and e the number of edges. We represent the vertices as $v_1, v_2, \ldots, v_n$, and we represent an edge between vertices x and y by $e_{x,y}$. We construct the edges in the following order, stopping as soon as e edges have been instantiated.

- (Step 1). The first $n-1$ edges are given by: $e_{1,2}, e_{1,3}, \ldots, e_{1,n}$.
- (Step 2). The next $n-2$ edges are given by: $e_{2,3}, e_{2,4}, \ldots, e_{2,n}$.
-
-
-
- (Step $n-1$). The last possible edge is given by: $e_{n-1,n}$.

Assume that we chose our e-th edge in step w. If $w = n-1$, then the graph is complete and is a clique in itself.

The more interesting case is when $w < n-1$. Then the graph has a clique of size $w+1$ (all vertices $v_1, v_2, \ldots, v_w, v_{w+1}$ are connected together), and $n-w-1$ singleton cliques each consisting of the remaining vertices $v_{w+2}, \ldots, v_n$. Thus the total number of cliques by this partitioning strategy equals $n-w$.

Furthermore, the number of edges in the graph is:

$$\begin{aligned} e &= (n-1)+(n-2)+\cdots+(n-w-1)+d \\ &= \frac{2wn-w^2+w-2n+2d}{2} \end{aligned} \tag{2}$$

where d is the number of edges chosen in step w. Since $n-w$ edges are listed in step w:

$$1 \le d \le n-w$$

Inserting the value of e in Equation (2) into Equation (1), we have:

$$\begin{aligned} c &= \left\lfloor \frac{1+\sqrt{4n^2-4n-8wn+4w^2+8n-4w-8d+1}}{2} \right\rfloor \\ &= \left\lfloor \frac{1+\sqrt{4(n-w)^2+4(n-w)+1-8d}}{2} \right\rfloor \end{aligned} \tag{3}$$

Equation (3) is a decreasing function of d. Thus the value of c for any d between the bounds of 1 and $n-w$ must lie between the values of c at these bounds.

For $d=1$,

$$\begin{aligned} c &= \left\lfloor \frac{1+\sqrt{(2(n-w)+1)^2-8}}{2} \right\rfloor \\ &= \left\lfloor \frac{1+\left\lfloor \sqrt{(2(n-w)+1)^2-8} \right\rfloor}{2} \right\rfloor \\ &= \left\lfloor \frac{1+(2(n-w)+1)-1}{2} \right\rfloor \\ &= n-w \end{aligned}$$

And, for $d=n-w$,

$$\begin{aligned} c &= \left\lfloor \frac{1+\sqrt{(2(n-w)-1)^2}}{2} \right\rfloor \\ &= n-w \end{aligned}$$

Thus the graph has exactly the number of cliques given by Theorem 1. ∎

By way of comparison, we list below the upper bound results for clique-partitioning that have been described in the literature, or that can be derived straightforwardly from equivalent graph-coloring results.

LEMMA 1. *A matching in G provides a value for θ_{upper}.*

PROOF. Consider all matched edges and the remaining independent vertices as cliques. ∎

This result is tight for bipartite graphs, and matchings can be obtained in polynomial time. However, for arbitrary graphs this result grossly overestimates θ_{upper}.

LEMMA 2. $\theta_{\text{upper}} = n - \delta$.

PROOF. We know that $\chi = \Delta + 1$ [1]. Now $\Delta(G') = n - \delta(G) - 1$. The theorem follows from Lemma 1. ∎

LEMMA 3.

$$\theta_{\text{upper}}(G) = n - \alpha(G') + 1 = n - \gamma(G) + 1$$

PROOF. From Syslo [2],

$$\chi \leq n - \alpha + 1$$

and $\alpha(G') = \gamma(G)$. ∎

Lemmas 1 through 3 furnish upper bounds that require such information as the degree of the graph, or the size of its maximum independent set. Our result, on the other hand, depends only on the numbers of the vertices and the edges in the graph—the two obvious parameters of the graph, and is therefore more general.

3. EVERY MAXIMAL CLIQUE IS PART OF SOME OPTIMAL CLIQUE-PARTITION

LEMMA 4. *For every non-maximal clique C_i in a clique-partition of size θ, an alternative clique-partition of size less than or equal to θ exists which contains a maximal clique C' such that every vertex in C_i is also in C'.*

PROOF. Let $C_1, C_2, \ldots, C_\theta$ be a clique-partition of size θ, and let C_i be any non-maximal clique in this clique-partition. Since C_i is not a maximal clique, there must be vertices in one or more other cliques in the clique-partition that are adjacent to all vertices in C_i. Augment C_i to form a maximal clique by borrowing these vertices from other cliques. The cliques from which vertices were borrowed stay as cliques (since the deletion of a vertex and its incident edges from a clique produces another clique), or become empty. Thus we obtain a clique-partition containing a maximal clique that includes all the vertices in C_i. If the vertex borrowing process produces some empty cliques, the size of the new clique-partition equals θ. ∎

Lemma 4 implies that every maximal clique is part of some optimal clique-partition. The lemma also suggests a procedure for clique-partitioning a graph: We can find an exact clique-partition of a graph by first selecting (non-deterministically) a maximal clique, then selecting (non-deterministically) a maximal clique in the remaining subgraph and so on. That is,

$$\theta(G) = \theta(G_{V-U}) + 1$$

where U is the set of nodes that form the maximal clique. This implies

$$\theta(G) = \min_{W \subseteq V} \{\theta(G_{V-W}) + 1\}$$

where W is one of the maximal cliques of G. Unfortunately, finding the maximal clique of a graph is also NP-complete [3].

4. PREVIOUS PARTITIONING AND COLORING ALGORITHMS

A number of optimal and sub-optimal coloring and partitioning algorithms have been described earlier. Some references are Brelaz [4], Wilkov [5], Tseng [6], and Leighton [7]. A variation of the clique-partitioning problem has been studied by Pullman [8], Pullman [9], and Rees [10], where a clique-partition is defined to be a set of cliques such that each edge in the graph is in exactly one clique. In the problem that we study, a partition is performed on the vertices and not on the edges. This problem has also been referred to as the *clique cover* problem [11]. We know that both clique-partitioning and coloring are *NP-Complete* problems [3]. Hence we will not discuss the earlier optimal algorithms—their computational complexity renders application infeasible.

4.1 Coloring Algorithms

A good comparison of the major coloring algorithms is given by Syslo [2], where four algorithms are considered. The INTERSEQCOLORING algorithm is a sequential coloring algorithm with interchanges in the colors allowed. In this algorithm, colors are assigned to an ordered list of vertices one at a time. Assume $v_1, v_2, \ldots, v_{i-1}$ have been colored with k colors and v_i is to be colored with $k+1$. This implies that v_i has neighbors in the colored set of vertices that have colors from 1 to k. Now if there exists complete subgraph on k vertices in the subgraph generated by

the neighbors of v_i, then the new color is necessary. Otherwise, it may be possible to interchange the colors of some neighbors of v_i, preserving the k colors of the subgraph $v_1, v_2, \ldots, v_{i-1}$ and free one of the first k colors for v_i. There are two variants of INTERSEQCOLORING; the order in which the vertices are processed can be largest-first (LF) or smallest-last (SL). The LF ordering orders vertices according to nonincreasing degree, $\deg(v_1) \geq \deg(v_2) \geq \cdots \geq \deg(v_n)$. The SL ordering $(v_1, v_2, \ldots, v_n)$ is

i. v_n is the minimum degree vertex of the given graph, G.
ii. For $i := n-1, n-2, \ldots, 2, 1$, v_i is a minimum degree vertex in the subgraph of G induced by $V - \{v_n, v_{n-1}, \ldots, v_{i+1}\}$.

The second algorithm is the DSATUR algorithm from Brelaz [4]. In this algorithm, the vertex with the maximal degree is assigned a color 1. Next, select a vertex with a maximal saturation degree. In case of a tie, pick any vertex of maximal degree in the uncolored subgraph and this vertex is colored with the least possible color. The third algorithm compared is the RLF algorithm [7]. This algorithm starts by assigning color 1 to the vertex with the maximal degree, say v_1. Once i vertices have been assigned to color 1, select a vertex from the set of uncolored vertices that are not adjacent to any colored vertices such that v_{i+1} has the maximum number of uncolored adjacent vertices that are adjacent to at least one colored vertex. Ties are broken by selecting the vertex that has the minimal degree among the set of uncolored vertices.

The fourth algorithm is the BACKTRACK algorithm which is a simple backtracking algorithm. It starts by assigning color 1 to the first vertex, say v_1, in an arbitrary ordering of vertices. The remaining vertices are colored consecutively by applying the following rule: The color for v_i is selected by finding the minimum color from the set of all feasible colors that can be assigned to v_i.

The conclusions reported in Syslo [2] were that INTERSEQCOLORING, DSATUR, and RLF algorithms generate solutions of almost the same quality with the last two algorithms performing considerably faster. The BACKTRACK algorithm does produce good results but at prohibitive computational expense.

Based on the reported conclusions, we compare the functional performance and the run-time performance of our clique-partitioning algorithms with the INTERSEQCOLORING algorithm. We used both orderings in our comparisons. Figure 1 shows a clique-partitioning algorithm using SL ordering. We label this the "COLOR-SL" clique-partitioning algorithm. The LF ordering INTERSEQCOLORING algorithm is labeled "COLOR-LF."

1. Given graph G, compute its complement graph G'.
2. Execute the INTERSEQCOLORING [2] using SL ordering on G'.
3. The chromatic number gives us the number of clique-partitions in G; the vertices with same chromatic index form a clique in G.

Figure 1. COLOR-SL clique-partitioning algorithm.

4.2 Tseng's Clique-Partitioning Algorithm

Recently, Tseng [6] has used clique-partitioning to solve some difficult problems in the register-transfer level design of processors. The problems he addresses are the allocation of registers, data operators and interconnection units. Tseng describes a new clique-partitioning algorithm which he uses for these allocation tasks. This algorithm is sketched out in Figure 2, and is hereafter referred to as "TSENG." The algorithm constructs maximal cliques one at a time. The edge that is best connected is used to initiate a clique that is then built upon until the clique is maximal. Only then is the next new clique initiated.

5. TWO NEW ALGORITHMS

In this section we describe two new algorithms that we have developed for clique-partitioning. The second one is derived from the first, the only modification being the addition of a tie-breaker rule. The basic algorithm is described in Figure 3.

METHOD-1: The first algorithm is identical to the basic algorithm in Figure 3, with the clarification that if more than one vertex is identified in Step 3, an arbitrary choice is made.

1. Pick the edge (p, q) which has the maximum number of common neighbors (a vertex is a common neighbor of an edge if it is connected to both vertices of the edge).
   If the graph has no edges, then *Stop*.
   If multiple edges have the same maximum number of common neighbors, then choose the edge which would result in the deletion of the fewest number of edges (see Step 3).
2. Cluster p and q into a clique.
3. Modify the graph by replacing p and q by a new vertex r. A vertex v is connected to r in the new graph if and only if v was connected to both p and q in the old graph. (All edges connecting p with a vertex to which q was not connected, and all edges connecting q with a vertex to which p was not connected, are deleted.)
4. If vertex r is isolated, *Goto* 1.
   Else pick an edge s which includes r as a vertex and which has the maximum number of common neighbors. If multiple edges meet this criterion, choose the edge which would result in the deletion of the fewest number of edges.
5. Rename r and s as p and q.
6. *Goto* 3.

In Steps 1 and 4, if multiple edges meet the conditions, an arbitrary choice is made.

Figure 2. TSENG clique-partitioning algorithm.

1. If all vertices have zero degree, goto Step *7*.
2. Choose a non-zero degree vertex with the smallest degree. Call this $N1$.
3. Of all the vertices connected to $N1$, choose the one with the smallest degree. Call this $N2$.
4. Cluster $N1$ and $N2$ into a new compound vertex $N3$.
5. Modify the graph by replacing vertices $N1$ and $N2$ with the new vertex $N3$. A vertex Nx is connected to $N3$ in the new graph iff Nx was connected to both $N1$ and $N2$ in the old graph. The degree of affected vertices are updated.
6. Goto Step *1*.
7. Any vertex not already clustered are made into singular clusters.

Figure 3. The basic algorithm.

METHOD-2: In Step 3 of the basic algorithm described in Figure 3, if more than one vertex is identified as a candidate for $N2$, we choose one that has a common neighbor with the edge ($N1$, $N2$). If no candidate has a common neighbor with ($N1$,$N2$) an arbitrary choice is made.

Unlike TSENG, our algorithms construct cliques in parallel. Furthermore, the *least* connected vertices are considered first. The algorithms are best understood by working on the example graph G illustrated in Figure 4(a). First, select a vertex with the smallest degree. Since G is a cubic graph, we pick any vertex for $N1$, say 1. Step 3 in Figure 3 distinguishes METHOD-1 from METHOD-2. In METHOD-1, vertex 5 might be picked as $N2$ since all vertices connected to vertex 1 have the same degree, whereas in METHOD-2, vertex 2 (or 3) will be selected as $N2$ since vertex 3 (or 2) is a common neighbor of (1,2) (or (1,3)). Continuing with METHOD-1, we cluster vertices 1 and 5 into a new vertex 7 that becomes $N3$. (For ease of exposition we number new vertices uniquely; in the implementation the number for the compound vertex $N3$ is the lesser of $N1$ and $N2$. Thus the graph does not grow during the partitioning.) The new modified graph $G1$ is shown in Figure 4(b). Edges (2,1) and (6,5) do not appear in $G1$ because neither vertex 2 nor vertex 6 is connected to both 1 and 5. We continue the next iteration of the algorithm on G_1 to yield the graph G_2 shown in Figure 4(c). Continuing on, we get the three cliques (1,5), (3,4) and (2,6). METHOD-2 would have yielded the cliques (1,2,3) and (4,5,6) which is an optimal clique-partition.

Timing complexity: We use an adjacency matrix to represent the graph. The degrees of vertices are explicitly represented in a one-dimensional array. Step 2 takes $O(n)$ time. Step 3 takes another $O(n)$. Step 5 takes $O(n)$ time since we need only AND two rows of the matrix. Iterating until all vertices are accounted for would take another $O(n)$ in the worst case yielding a net time complexity for METHOD-1 of $O(n^2)$. The complexity of Step 3 for METHOD-2

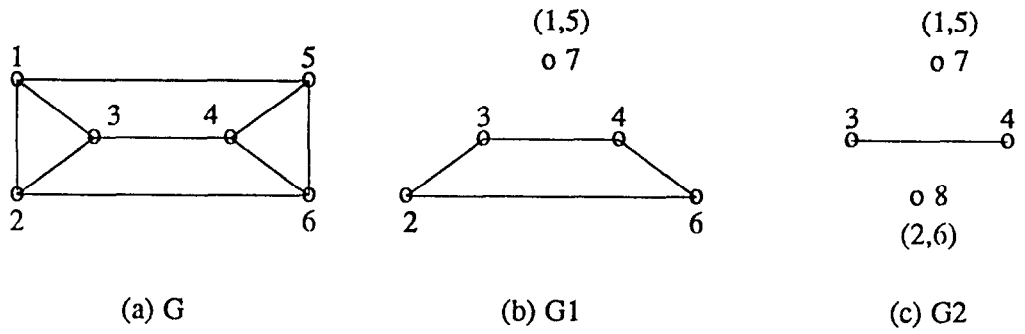

Figure 4. An example of clique-partitioning.

algorithm is $O(n^2)$ since in the worst case all candidate $(n-1)$ $N2$'s will have to be checked for a common neighbor. Thus the net worst case time complexity for METHOD-2 is $O(n^3)$. The complexity of Step 3 for METHOD-2 can be reduced to $O(e)$ if adjacency lists are used to determine the common neighbor. But this will increase the complexity of Step 5.

6. PERFORMANCE COMPARISONS

We have implemented the five algorithms COLOR-SL, COLOR-LF, TSENG, METHOD-1 and METHOD-2 in Pascal. The code for the coloring algorithms was adapted from Syslo [2]. Our implementation of Tseng's algorithm does not use the data structures that he uses; instead of representing the graph by a structured list of edges, we use an adjacency matrix representation. Our implementation is certainly more wasteful of memory, but it is probably faster. The run-time information for the coloring algorithms includes the time spent complementing the original graph before running the coloring algorithm.

We ran all the five algorithms on a set of small graphs taken from the literature. Most of these have been used for bench marking coloring algorithms. The graphs used were:

cex_pull82 : a 12 vertices, 30 edges example from Pullman [9]
cex_pull84 : a 10 vertices, 18 edges example form Pullman [8]
cex_tseng1 : an 8 vertices, 14 edges example from Tseng [6]
cex_tseng2 : a 10 vertices, 9 edges example form Tseng [6]
cex_wilk1 : a 10 vertices, 17 edges example from Wilkov [5]
cex_wilk2 : a 20 vertices, 40 edges example from Wilkov [5]
cex_leig : an 11 vertices, 26 edges example from Leighton [7]
cex_matu : a 34 vertices, 194 edges example from Matula [12]
cex_m3 : the M3 Mycielski's graph
cex_m4 : the M4 Mycielski's graph
cex_m5 : the M5 Mycielski's graph

Mycielski's graphs are defined in Ore [13]. Table 1 shows the results of this experiment. With a couple of exceptions, all algorithms perform optimally. TSENG is suboptimal on "cex_matu," and only METHOD-2 and COLOR-LF are optimal on "cex_wilk2."

For further comparison, we used five sets of ten randomly generated graphs. All graphs had 50 vertices. The number of edges in a graph was constant within a set, but varied across sets. Table 2 shows the results obtained. The entries in the table correspond to the number of cliques obtained and each column corresponds to one random graph. Since the differences in numbers of cliques are small on an absolute scale, we compared performance by computing the percent deviation of each result obtained from the optimal. The optimal value for a column was taken to be the lowest number of cliques in that column. For example, in set 1 and column 6, 25 is taken to be the optimal number of cliques for this column since it is the lowest and thus METHOD-1 and

Table 1. Performance of algorithms on small graphs. Entries denote number of cliques.

Instance	METHOD-1	METHOD-2	COLOR-SL	COLOR-LF	TSENG
cex_pull82	4	4	4	4	4
cex_pull84	6	6	6	6	6
cex_tseng1	4	4	4	4	4
cex_tseng2	6	6	6	6	6
cex_wilk1	3	3	3	3	3
cex_wilk2	4	3	4	3	4
cex_leig	5	5	5	5	5
cex_matu	8	8	8	8	9
cex_m3	3	3	3	3	3
cex_m4	4	4	4	4	4
cex_m5	5	5	5	5	5

METHOD-2 have 0% deviation, whereas COLOR-SL, COLOR-LF and TSENG have deviations of 4%, 4% and 12%, respectively. The percent deviation in the last column is obtained by computing the average of all deviations in a particular row, i.e., adding up the deviations and dividing by 10.

Table 2. Performance of algorithms on five sets of 10 random graphs. Entries denote number of cliques; each column corresponds to one random graph.

Set 1	vertices=50, edges=100										% deviation
METHOD-1	25	27	25	27	25	25	27	26	26	26	0.0
METHOD-2	25	27	25	27	25	25	27	26	26	26	0.0
COLOR-SL	27	28	27	29	26	26	28	27	26	26	4.26
COLOR-LF	27	29	25	28	26	26	28	26	27	26	3.46
TSENG	28	29	28	30	29	28	29	33	29	31	13.46
Set 2	vertices=50, edges=250										% deviation
METHOD-1	19	19	20	19	21	19	20	19	18	20	2.16
METHOD-2	19	19	20	19	19	19	20	19	18	19	0.55
COLOR-SL	19	19	21	19	21	19	21	21	18	22	5.32
COLOR-LF	19	19	20	20	21	19	20	19	18	18	1.57
TSENG	20	19	21	19	20	20	21	19	19	22	5.33
Set 3	vertices=50, edges=500										% deviation
METHOD-1	14	14	14	14	13	13	13	13	13	14	0.76
METHOD-2	13	14	14	14	13	13	14	13	13	14	0.76
COLOR-SL	14	15	14	15	14	15	15	13	14	14	6.78
COLOR-LF	14	15	14	15	14	14	14	13	13	14	4.46
TSENG	13	15	14	14	14	14	16	14	14	14	6.06
Set 4	vertices=50, edges=750										% deviation
METHOD-1	9	10	10	9	10	10	9	10	11	10	7.77
METHOD-2	9	10	10	10	9	10	9	9	10	9	4.44
COLOR-SL	9	9	11	9	10	10	9	10	9	9	4.33
COLOR-LF	9	10	10	9	9	9	9	9	10	10	3.33
TSENG	10	10	11	10	11	10	10	10	11	11	14.33
Set 5	vertices=50, edges=1000										% deviation
METHOD-1	6	6	7	6	7	6	6	7	6	6	3.32
METHOD-2	6	6	8	6	6	6	6	7	6	6	3.33
COLOR-SL	6	6	6	6	6	6	6	7	6	6	0.0
COLOR-LF	6	6	7	6	6	6	6	7	6	6	1.66
TSENG	7	7	7	6	7	7	7	8	7	7	14.71

The results in Table 2 indicate that METHOD-1 and METHOD-2 have the least percent deviation from optimal, with TSENG doing the worst. However, as the connectivity is increased, COLOR-SL and COLOR-LF become comparable, and eventually better than METHOD-1 and METHOD-2. This is clearly plausible since a large connectivity graph for clique-partitioning is a sparse graph for coloring and any good coloring algorithm will perform close to optimally.

Table 3. Performance of algorithms with varying connectivities. Entries denote average number of cliques obtained over 10 random graphs.

10 vertices	% connectivity									% deviation
	10	20	30	40	50	60	70	80	90	
METHOD-1	7.5	6.0	5.1	4.4	4.0	3.5	3.1	2.9	2.2	1.43
METHOD-2	7.5	6.0	5.1	4.4	4.0	3.5	3.1	2.9	2.1	0.90
COLOR-SL	7.5	6.0	5.2	4.4	4.1	3.4	3.0	2.9	2.1	0.72
COLOR-LF	7.5	6.0	5.0	4.4	4.0	3.5	3.1	2.9	2.1	0.68
TSENG	7.5	6.0	5.6	4.8	4.1	3.7	3.2	3.0	2.1	4.67
50 vertices	% connectivity									% deviation
	10	20	30	40	50	60	70	80	90	
METHOD-1	24.4	19.2	15.6	13.6	11.4	10.0	8.2	6.8	5.0	2.64
METHOD-2	24.4	19.0	16.2	13.4	11.4	9.8	8.5	6.6	5.0	2.60
COLOR-SL	25.4	20.2	16.6	13.8	12.0	10.0	8.3	6.4	4.7	3.88
COLOR-LF	24.8	19.8	16.6	14.0	11.2	9.6	8.0	6.5	4.7	2.01
TSENG	27.2	19.8	16.6	14.4	12.6	10.5	9.3	7.5	5.5	11.10
100 vertices	% connectivity									% deviation
	10	20	30	40	50	60	70	80	90	
METHOD-1	42.2	33.3	27.4	22.8	18.8	15.6	12.9	10.1	7.1	0.86
METHOD-2	41.8	33.1	26.9	22.8	18.8	15.6	13.1	10.1	6.9	0.31
COLOR-SL	44.4	35.0	28.1	23.3	19.0	15.9	13.4	10.4	6.9	3.10
COLOR-LF	43.1	34.0	28.0	22.9	18.7	15.9	12.8	10.2	7.1	1.74
TSENG	44.7	35.0	29.3	24.5	20.3	17.6	15.1	11.7	8.6	12.00

To verify that these results held over a wider range of graphs, we tested all algorithms on graphs of 10 and 100 vertices, varying the number of edges for each case between 10 and 90 percent of full connectivity. Table 3 shows the results. The entries denote the average number of cliques obtained over ten random graphs. For example, with ten random graphs with 100 vertices and 40% connectivity, 22.8 number of cliques were obtained on an average by METHOD-1. The percent deviation was calculated in a similar manner as in Table 2. For 10 vertices, the algorithms seem to perform well, although TSENG does somewhat poorly in the middle range of connectivity. For 50 vertices, our algorithms perform the best until the high connectivity range where the coloring algorithms, for reasons discussed above, start doing better. The same trend seems to hold for 100 vertex graphs. However, the difference between the coloring algorithms and ours at the high connectivity end is significantly smaller. It appears that as the size of the graph increases, our algorithms perform increasingly better relative to the coloring algorithms.

Results in Table 3 also indicate that METHOD-2 performs better, albeit marginally, than METHOD-1; that TSENG produces the worst results; and that among the coloring algorithms, COLOR-LF performs better than COLOR-SL.

Figure 5 shows a plot of the performance of METHOD-1, METHOD-2, COLOR-SL and COLOR-LF algorithms with $n = 100$ vertices and with varying connectivities.

Table 4 shows the run-times of the various algorithms obtained when executed on an Apollo DN330 workstation. We see that METHOD-1 and METHOD-2 are fast algorithms. Both the coloring algorithms are slow because of the fact that they have to complement the graph before coloring can proceed. The run-times for TSENG are for our implementation, which uses different data structures than what was originally proposed. We believe, however, that our implemenation is faster. The plot in Figure 6 illustrates the time complexity superiority of our algorithms. The curves for METHOD-1 and METHOD-2 are overlapped since they are very close to each other.

Table 4. CPU run-times (in seconds).

30 vertices	% connectivity					
	10	20	30	40	50	60
METHOD-1	.027	.033	.034	.037	.034	.040
METHOD-2	.028	.034	.035	.038	.040	.042
COLOR-SL	.408	.235	.145	.115	.084	.047
COLOR-LF	.344	.246	.163	.108	.083	.060
TSENG	.086	.132	.160	.205	.205	.214
40 vertices	% connectivity					
	10	20	30	40	50	60
METHOD-1	.047	.054	.058	.062	.065	.068
METHOD-2	.048	.055	.061	.065	.066	.069
COLOR-SL	.751	.517	.351	.213	.142	.095
COLOR-LF	.753	.631	.349	.260	.166	.119
TSENG	.217	.325	.440	.488	.532	.532
50 vertices	% connectivity					
	10	20	30	40	50	60
METHOD-1	.071	.083	.091	.095	.100	.104
METHOD-2	.072	.085	.092	.097	.102	.105
COLOR-SL	1.68	.992	.700	.399	.291	.186
COLOR-LF	1.76	1.03	.665	.443	.302	.189
TSENG	.421	.736	.967	1.06	1.093	.970

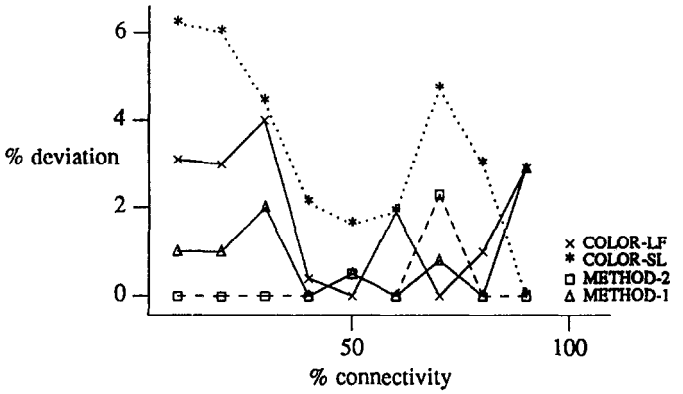

Figure 5. Plot of performance with 100 vertices and varying connectives.

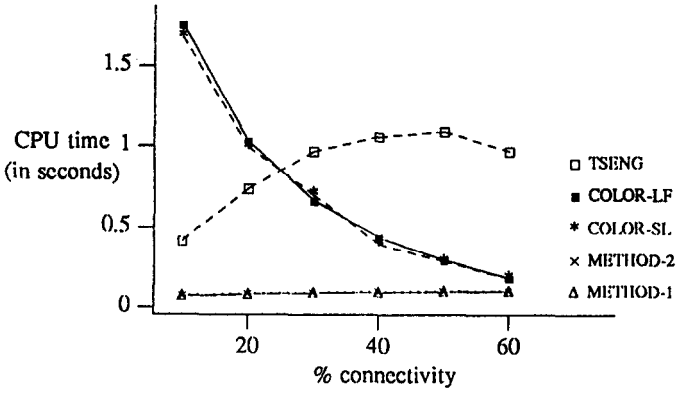

Figure 6. CPU run-times with 50 vertices and varying connectives.

The sparser the graph, the faster the algorithm execution. Since the coloring algorithms are operating on the complement of the graph, their run-times decrease with increasing connectivities.

7. SUMMARY

In this paper, we have reported some new results in clique-partitioning. Our theoretical investigations have resulted in a new upper bound for clique-partitioning and a proof that every maximal clique is part of some other clique-partition.

We have also developed two new clique-partitioning algorithms. We have tested these extensively against two well-known coloring algorithms and against a recently published clique-partitioning algorithm, with consistently favorable results. In particular, it appears that our algorithms perform increasingly better relative to the coloring algorithms for larger graphs.

Although both clique-partitioning and coloring are important problems, coloring has been by far the most intensively researched. One conclusion of this research is that one should not always use a coloring algorithm to solve a clique-partitioning problem; it is usually better to use a clique-partitioning algorithm itself.

REFERENCES

1. M.N.S. Swamy and K. Thulasiraman, *Graphs, Networks, and Algorithms*, John Wiley and Sons, (1981).
2. M.M. Syslo, N. Deo and J.S. Kowalik, *Discrete Optimization Algorithms*, Prentice Hall Inc., Englewood Cliffs, NJ, (1984).
3. M.S. Garey and D.S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*, Freeman and Co., San Francisco, (1979).
4. D. Brelaz, New methods to color the vertices of a graph, *Comm. ACM* **22**, 251–256 (1979).
5. R.S. Wilkov and W.H. Kim, A practical approach to the chromatic partition problem, *J. of the Franklin Institute* **289**, 333–349 (May 1970).
6. C.J. Tseng, Automated synthesis of data paths in digital systems, Ph. D. Thesis, Dept. of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburg, PA, (April 1984).
7. F.T. Leighton, A graph coloring algorithm for large scheduling problems, *J. Res. Nat. Bur. Standards* **84**, 489–506 (1979).
8. N.J. Pullman, Clique covering of graphs IV: Algorithms, *SIAM J. Computing* **13**, 57–75 (Feb 1984).
9. N.J. Pullman *et al.*, Clique coverings of graphs V: maximal-clique-partitions, *Bull. Austral. Math. Soc.* **25**, 337–356 (1982).
10. R. Rees, Minimal clique-partitions and pairwise balanced designs, *Discrete Mathematics 61*, 269–280 (1986).
11. R.M. Karp, Reducibility among combinational problems, In *Complexity of Computer Computations*, (R.E. Miller and J.W. Thatcher, Eds.), pp. 85–103, Plenum Press, New York and London, (1972).
12. D.W. Matula, G. Marble and J.D. Isaacson, Graph coloring algorithms, In *Graph Theory and Computing* (Edited by R.C. Read), pp. 109–122, Academic Press, London, (1972).
13. O. Ore, *The Four Color Problem*, Academic Press, New York, (1967).